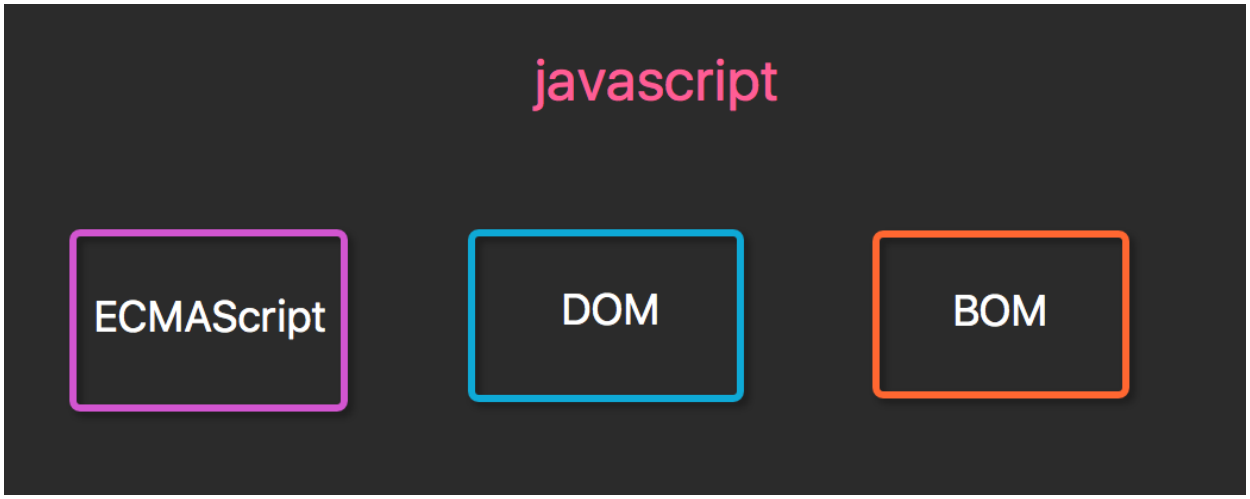
知识预览

- BOM对象
- DOM对象(DHTML)
- 8 实例练习

- 1992年Nombas开发出C-minus-minus(C--)的嵌入式脚本语言(最初绑定在CEnvi软件中).后将其改名ScriptEase.(客户端执行的语言)
- Netscape(网景)接收Nombas的理念,(Brendan Eich)在其Netscape Navigator 2.0产品中开发出一套livescript的脚本语言.Sun和Netscape共同完成.后改名叫Javascript
- 微软随后模仿在其IE3.0的产品中搭载了一个JavaScript的克隆版叫Jscript.
- 为了统一三家,ECMA(欧洲计算机制造协会)定义了ECMA-262规范.国际标准化组织及国际电工委员会（ISO/IEC）也采纳 ECMAScript 作为标准（ISO/IEC-16262）。从此，Web 浏览器就开始努力（虽然有着不同的程度的成功和失败）将 ECMAScript 作为 JavaScript 实现的基础。EcmaScript是规范.

尽管 ECMAScript 是一个重要的标准，但它并不是 JavaScript 唯一的部分，当然，也不是唯一被标准化的部分。实际上，一个完整的 JavaScript 实现是由以下 3 个不同部分组成的：

- 核心（ECMAScript）
- 文档对象模型（DOM） Document object model (整合js，css，html)
- 浏览器对象模型（BOM） Broswer object model（整合js和浏览器）
- Javascript 在开发中绝大多数情况是基于对象的.也是面向对象的.



简单地说，ECMAScript 描述了以下内容：

- 语法
- 类型
- 语句
- 关键字
- 保留字
- 运算符
- 对象 (封装 继承 多态) 基于对象的语言.使用对象.

```
{#1 直接编写#}
alert('hello yuan')
</script>
{#2 导入文件#}
="hello.js">
</script>
```

x=5y=6z=x+y

在代数中，我们使用字母（比如 x）来保存值（比如 5）。

通过上面的表达式 z=x+y，我们能够计算出 z 的值为 11。

在 JavaScript 中，这些字母被称为变量。

0 变量是弱类型的(很随便)；

1 声明变量时不用声明变量类型. 全都使用var关键字;

```
var a;
```

2一行可以声明多个变量.并且可以是不同类型.

```
var name="yuan",
age=20,
job="lecturer";
```

3(了解) 声明变量时 可以不用var. 如果不用var 那么它是全局变量.

4变量命名,首字符只能是字母,下划线,$美元符 三选一，且区分大小写，x与X是两个变量

5 变量还应遵守以下某条著名的命名规则：

Camel 标记法首字母是小写的，接下来的字母都以大写字符开头。例如：var myTestValue = 0, mySecondValue = "hi";Pascal 标记法首字母是大写的，接下来的字母都以大写字符开头。例如：Var MyTestValue = 0, MySecondValue = "hi";匈牙利类型标记法在以 Pascal 标记法命名的变

量前附加一个小写字母（或小写字母序列），说明该变量的类型。例如，i 表示整数，s 表示字符串，如下所示"Var iMyTestValue = 0, sMySecondValue = "hi";

注意：

View Code

1　每行结束可以不加分号．没有分号会以换行符作为每行的结束

View Code

2　注释 支持多行注释和单行注释．/* */　　//

3使用{}来封装代码块

常量 ：直接在程序中出现的数据值

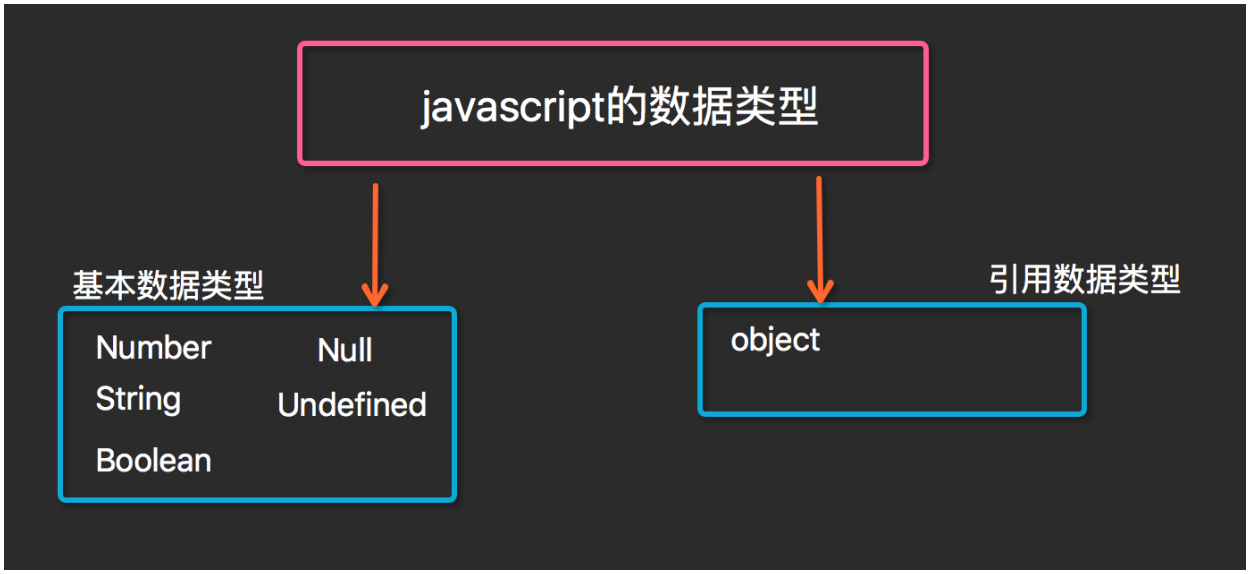
标识符：

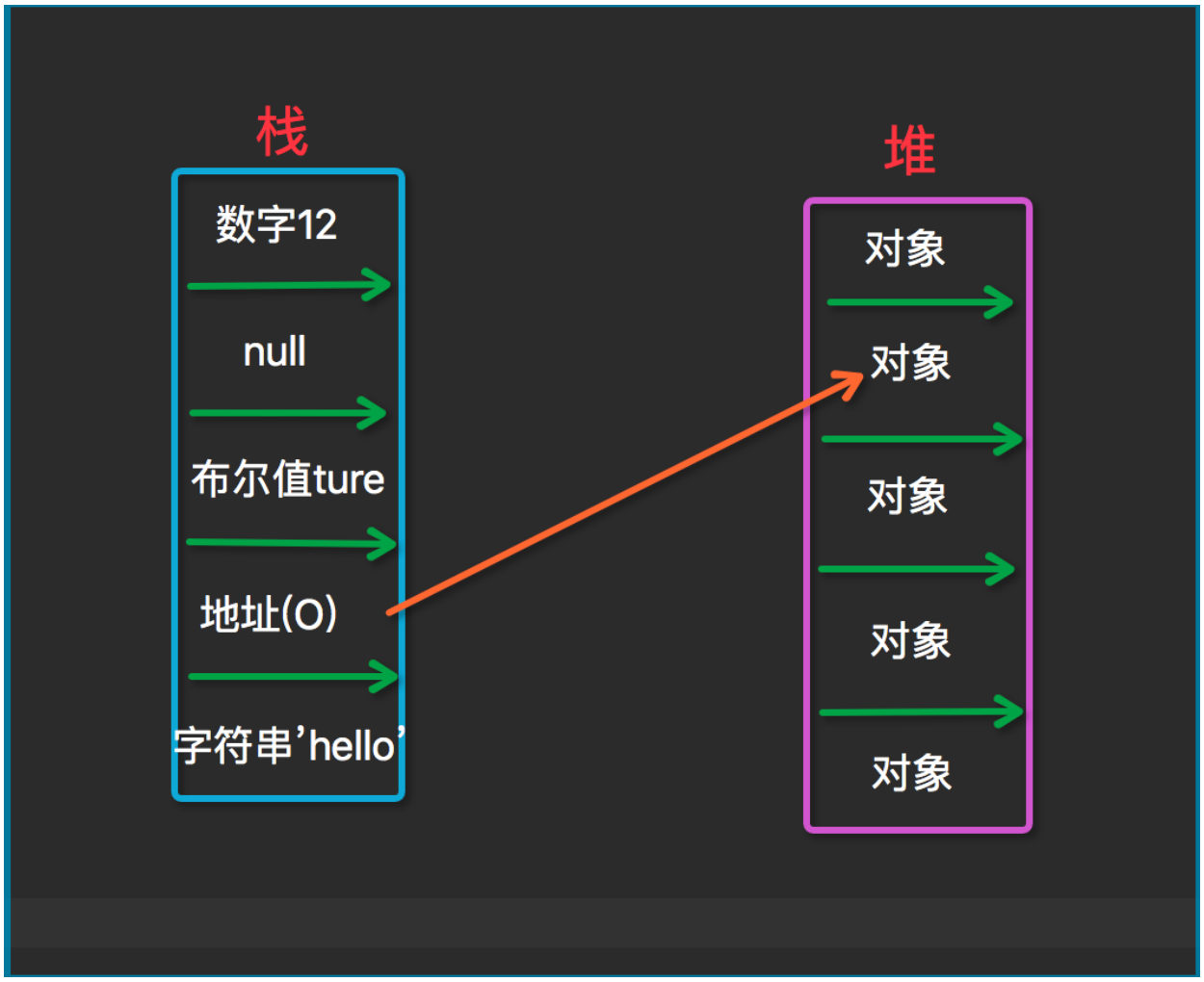
1. 由不以数字开头的字母、数字、下划线(_)、美元符号($)组成
2. 常用于表示函数、变量等的名称
3. 例如：_abc,$abc,abc,abc123是标识符，而1abc不是
4. JavaScript语言中代表特定含义的词称为保留字，不允许程序再定义为标识符

- ECMA v3标准保留的JavaScript的关键字

break	case	catch	continue	default
delete	do	else	false	finally
for	function	if	in	instanceof
new	null	return	switch	this
throw	true	try	typeof	var
void	while	with		


自定义标识符
字符串常量
保留字
<script>
function sayHello(){
var hellostr="您好";
document.write(hellostr);
}
sayHello();
</script>


javascript的数据类型
基本数据类型
Number
String
Boolean
Null
Undefined
引用数据类型
object



数字类型(Number)

简介最基本的数据类型不区分整型数值和浮点型数值所有数字都采用64位浮点格式存储，相当于Java和C语言中的double格式能表示的最大值是±1.7976931348623157 x 10308 能表示的最小值是±5 x 10-324

整数：

在JavaScript中10进制的整数由数字的序列组成

精确表达的范围是-9007199254740992 (-253) 到 9007199254740992 (253)

超出范围的整数，精确度将受影响

浮点数：

使用小数点记录数据

例如：3.4，5.6

使用指数记录数据

例如：4.3e23 = 4.3 x 1023

16进制和8进制数的表达

16进制数据前面加上0x，八进制前面加0

16进制数是由0-9,A-F等16个字符组成

8进制数由0-7等8个数字组成

16进制和8进制与2进制的换算

2进制: 1111 0011 1101 0100 <-----> 16进制:0xF3D4 <-----> 10进制:62420# 2进制: 1 111 001 111 010 100 <-----> 8进制:0171724

字符串(String)

简介是由Unicode字符、数字、标点符号组成的序列字符串常量首尾由单引号或双引号括起JavaScript中没有字符类型常用特殊字符在字符串中的表达字符串中部分特殊字符必须加上右划线\常用的转义字符

\n:换行 \':单引号 \":双引号 \\:右划线

String数据类型的使用

- 特殊字符的使用方法和效果
- Unicode的插入方法

```
var str="\u4f60\u597d\n欢迎来到\"JavaScript世界\"";
alert(str);
</script>
```

布尔型(Boolean)

简介Boolean类型仅有两个值：true和false，也代表1和0，实际运算中true=1,false=0布尔值也可以看作on/off、yes/no、1/0对应true/falseBoolean值主要用于JavaScript的控制语句，例如

```
if (x==1){ y=y+1; }else { y=y-1; }
```

Null & Undefined

Undefined 类型

Undefined 类型只有一个值，即 undefined。当声明的变量未初始化时，该变量的默认值是 undefined。

当函数无明确返回值时，返回的也是值 "undefined";

Null 类型

另一种只有一个值的类型是 Null，它只有一个专用值 null，即它的字面量。值 undefined 实际上是从值 null 派生来的，因此 ECMAScript 把它们定义为相等的。

尽管这两个值相等，但它们的含义不同。undefined 是声明了变量但未对其初始化时赋予该变量的值，null 则用于表示尚未存在的对象（在讨论 typeof 运算符时，简单地介绍过这一点）。如果函数或方法要返回的是对象，那么找不到该对象时，返回的通常是 null。

```
var person=new Person()
var person=null
```

数据类型转换

JavaScript属于松散类型的程序语言变量在声明的时候并不需要指定数据类型变量只有在赋值的时候才会确定数据类型表达式中包含不同类型数据则在计算过程中会强制进行类别转换

数字 + 字符串：数字转换为字符串

数字 + 布尔值：true转换为1，false转换为0

字符串 + 布尔值：布尔值转换为字符串true或false

强制类型转换函数

函数parseInt：　强制转换成整数　例如parseInt("6.12")=6 ; parseInt("12a")=12 ; parseInt("a12")=NaN ;parseInt("1a2")=1函数parseFloat：　强制转换成浮点数

parseFloat("6.12")=6.12函数eval：　将字符串强制转换为表达式并返回结果

eval("1+1")=2 ; eval("1<2")=true

类型查询函数(typeof)

ECMAScript 提供了 typeof 运算符来判断一个值是否在某种类型的范围内。可以用这种运算符判断一个值是否表示一种原始类型：如果它是原始类型，还可以判断它表示哪种原始类型。

函数typeof ：查询数值当前类型 (string / number / boolean / object)

例如typeof("test"+3)　"string"例如typeof(null)　"object "例如

typeof(true+1)　"number"例如typeof(true-false)　"number"

加(+)、 减(-)、 乘(*) 、除(/) 、余数(%) 加、减、乘、除、余数和数学中的运算方法一样 例如：9/2=4.5，4*5=20，9%2=1

-除了可以表示减号还可以表示负号 例如：x=-y

+除了可以表示加法运算还可以用于字符串的连接 例如："abc"+"def"="abcdef"

递增(＋＋) 、递减(－－)

假如x=2，那么x++表达式执行后的值为3，x--表达式执行后的值为1

i++相当于i=i+1，i--相当于i=i-1

递增和递减运算符可以放在变量前也可以放在变量后：--i

```
var i=1;
console.log(i++);
console.log(++i);
console.log(i--);
console.log(--i);
```

一元加减法：

```
var a=1;
var b=1;
a=-a;  //a=-1

var c="10";
alert(typeof (c));
c=+c;    //类型转换
```

```
    alert(typeof (c));
//    -------------------
    var d="yuan";
    d=+d;
    alert(d);//NaN:属于Number类型的一个特殊值,当遇到将字符串转成数字无效时,就会得到一个NaN数据
    alert(typeof(d));//Number

    //NaN特点:

    var n=NaN;

    alert(n>3);
    alert(n<3);
    alert(n==3);
    alert(n==NaN);

    alert(n!=NaN);//NaN参与的所有的运算都是false,除了!=
```

等于（==）、不等于（!=）、大于（>）、小于（<）大于等于(>=)、小于等于(<=)与(&&)、或(||)、非(!)

```
1 && 1 = 1  1 || 1 = 1
1 && 0 = 0  1 || 0 = 1
0 && 0 = 0  0 || 0 = 0

!0=1
!1=0
```

逻辑 AND 运算符(&&)

逻辑 AND 运算的运算数可以是任何类型的，不止是 Boolean 值。

如果某个运算数不是原始的 Boolean 型值，逻辑 AND 运算并不一定返回 Boolean 值：

- 如果某个运算数是 null，返回 null。
- 如果某个运算数是 NaN，返回 NaN。
- 如果某个运算数是 undefined，返回undefined。

逻辑 OR 运算符(||)

与逻辑 AND 运算符相似，如果某个运算数不是 Boolean 值，逻辑 OR 运算并不一定返回 Boolean 值

赋值 = JavaScript中=代表赋值，两个等号==表示判断是否相等例如，x=1表示给x赋值为1if (x==1){...}程序表示当x与1相等时if(x=="on"){…}程序表示当x与"on"相等时 配合其他运算符形成的简化表达式例如i+=1相当于i=i+1，x&=y相当于x=x&y

实例：

View Code

执行类型转换的规则如下：

- 如果一个运算数是 Boolean 值，在检查相等性之前，把它转换成数字值。false 转换成 0，true 为 1。
- 如果一个运算数是字符串，另一个是数字，在检查相等性之前，要尝试把字符串转换成数字。
- 如果一个运算数是对象，另一个是字符串，在检查相等性之前，要尝试把对象转换成字符串。
- 如果一个运算数是对象，另一个是数字，在检查相等性之前，要尝试把对象转换成数字。

在比较时，该运算符还遵守下列规则：

- 值 null 和 undefined 相等。
- 在检查相等性时，不能把 null 和 undefined 转换成其他值。
- 如果某个运算数是 NaN，等号将返回 false，非等号将返回 true。
- 如果两个运算数都是对象，那么比较的是它们的引用值。如果两个运算数指向同一对象，那么等号返回 true，否则两个运算数不等。

表达式	值
null == undefined	true
"NaN" == NaN	false
5 == NaN	false
NaN == NaN	false
NaN != NaN	true
false == 0	true
true == 1	true
true == 2	false
undefined == 0	false
null == 0	false
"5" == 5	true

```
varbResult = "Blue"< "alpha";
alert(bResult); //输出 true
```

在上面的例子中，字符串 "Blue" 小于 "alpha"，因为字母 B 的字符代码是 66，字母 a 的字符代码是 97。

比较数字和字符串

另一种棘手的状况发生在比较两个字符串形式的数字时，比如：

```
varbResult = "25"<"3";
alert(bResult); //输出 "true"
```

上面这段代码比较的是字符串 "25" 和 "3"。两个运算数都是字符串，所以比较的是它们的字符代码（"2" 的字符代码是 50，"3" 的字符代码是 51）。

不过，如果把某个运算数该为数字，那么结果就有趣了：

```
varbResult = "25"<3;
alert(bResult); //输出 "false"
```

这里，字符串 "25" 将被转换成数字 25，然后与数字 3 进行比较，结果不出所料。

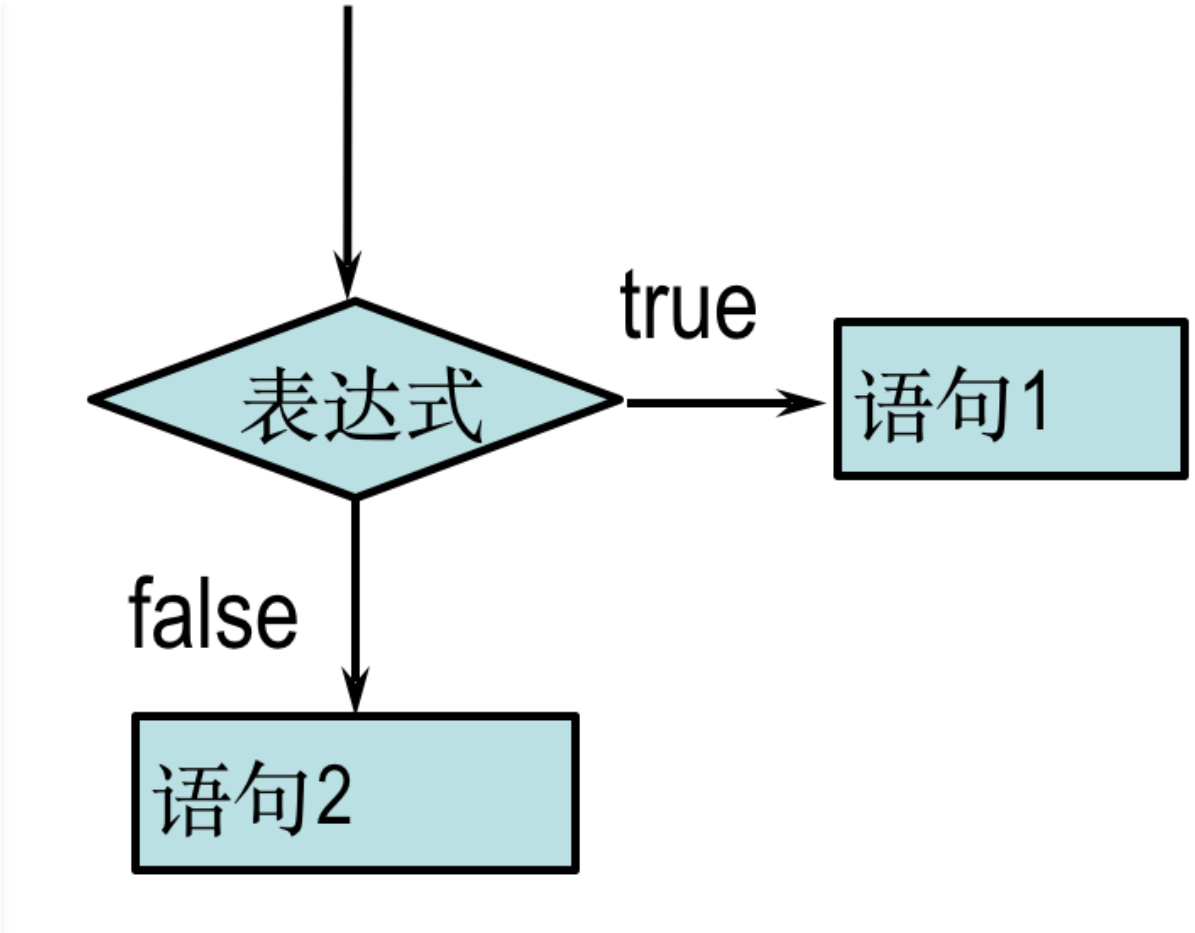
总结：

```
比较运算符两侧如果一个是数字类型,一个是其他类型,会将其类型转换成数字类型.比较运算符两侧如果都是字符串类型,比较的是最高位的asc码,如果最高位相等,继续取第二位比较.
```

```
var temp=new Object();// false;[];0; null; undefined;object(new Object();)if(temp){        console.log("yuan")    }else {console.log("alex")    }
```

等号和非等号的同类运算符是全等号和非全等号。这两个运算符所做的与等号和非等号相同，只是它们在检查相等性前，不执行类型转换。

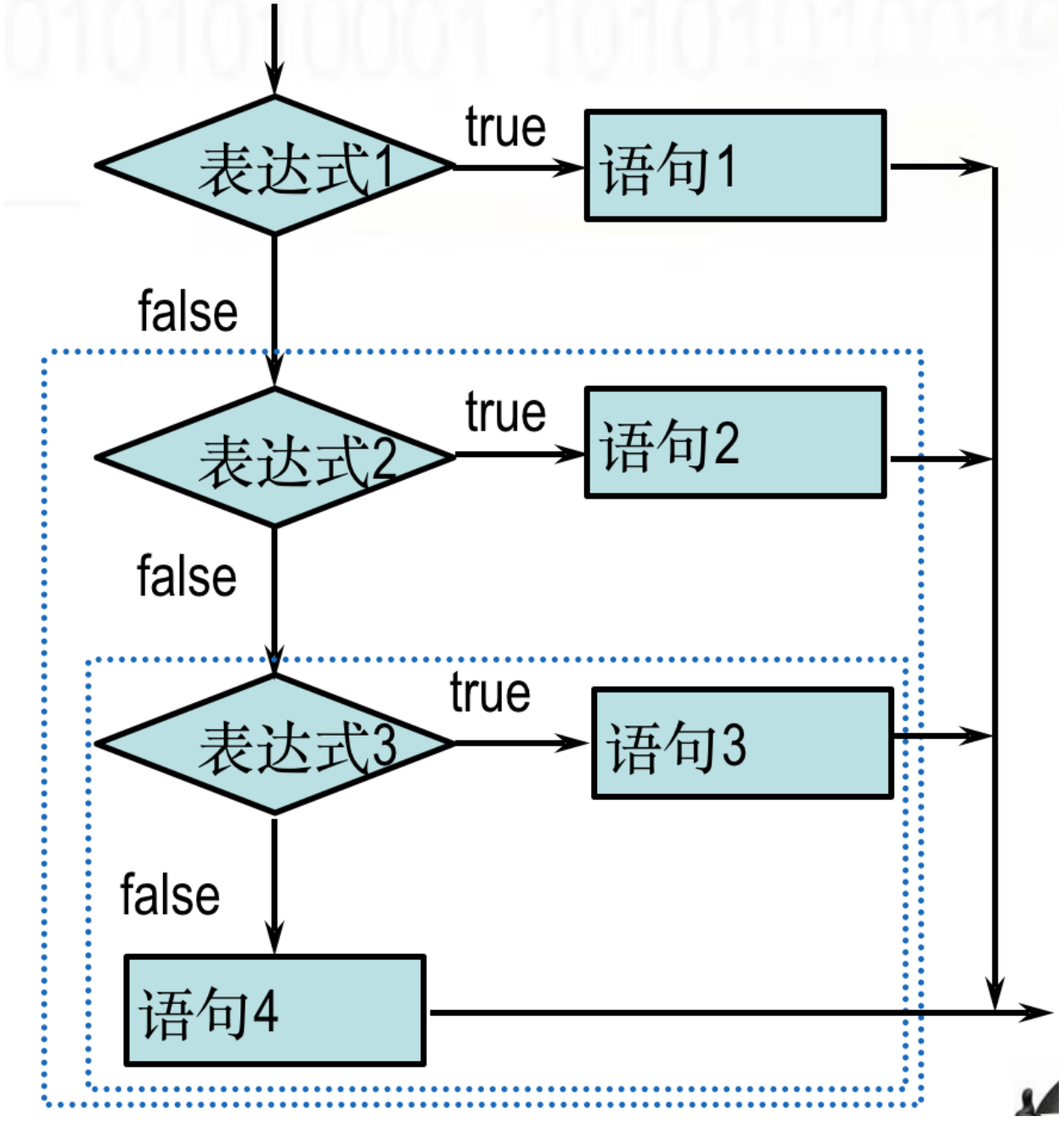
if-else基本格式if （表达式）{语句１；.....}else{语句２；.....}功能说明如果表达式的值为true则执行语句1,否则执行语句2




```
var x= (new Date()).getDay();//获取今天的星期值，0为星期天var y;if ( (x==6) || (x==0) ) {y="周末";}else{y="工作日";}alert(y);//等价于y="工作日";if ( (x==6) || (x==0) ) {y="周末";}
```

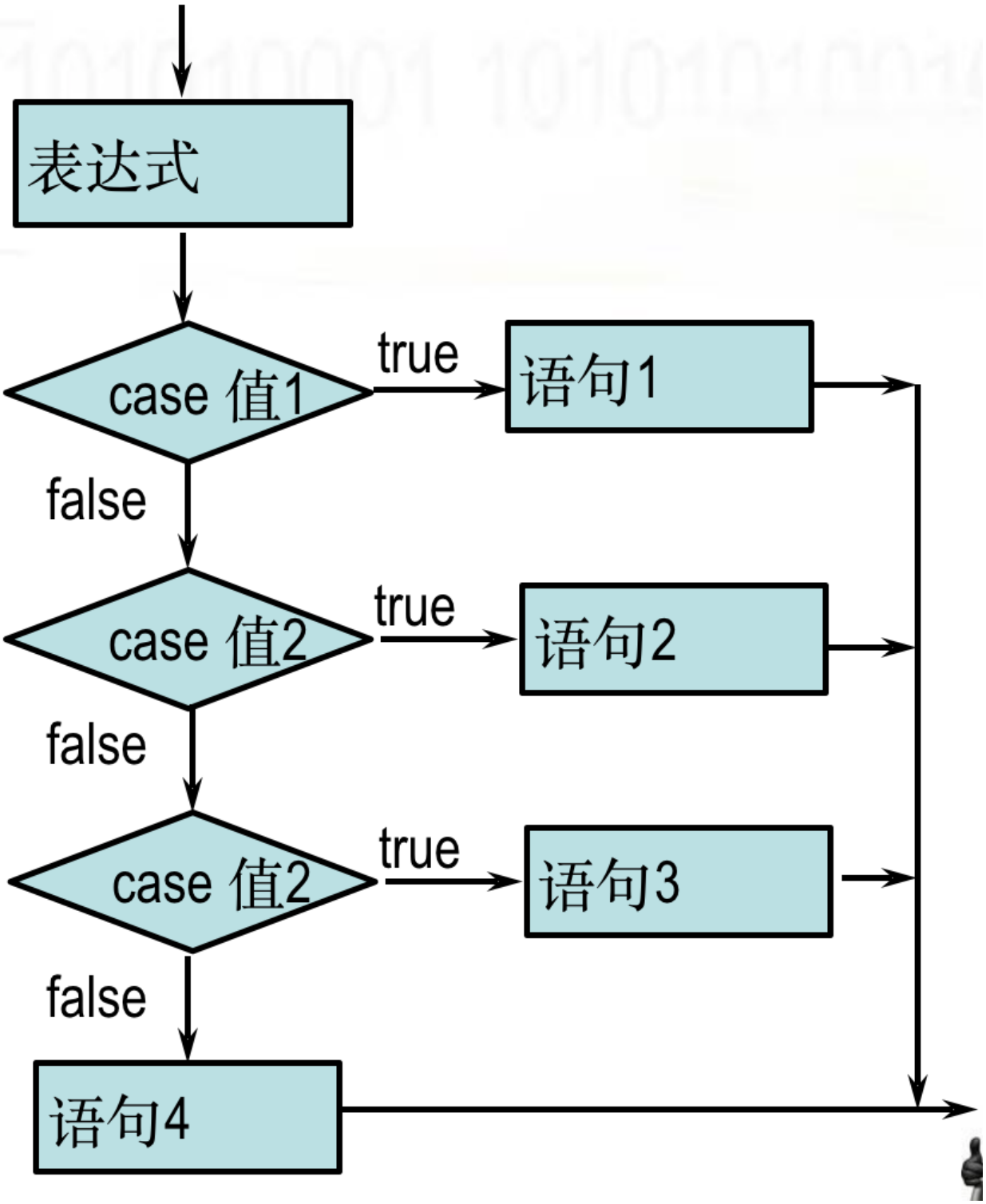
if 可以单独使用

if语句嵌套格式if （表达式1） {　　语句1;}elseif （表达式2）{　　语句2;}elseif （表达式3）{　　语句3;} else{　　语句4;}



View Code

switch基本格式switch （表达式） {　　case 值1:语句1;break;　　case 值2:语句2;break;　　case 值3:语句3;break;　　default:语句4;}

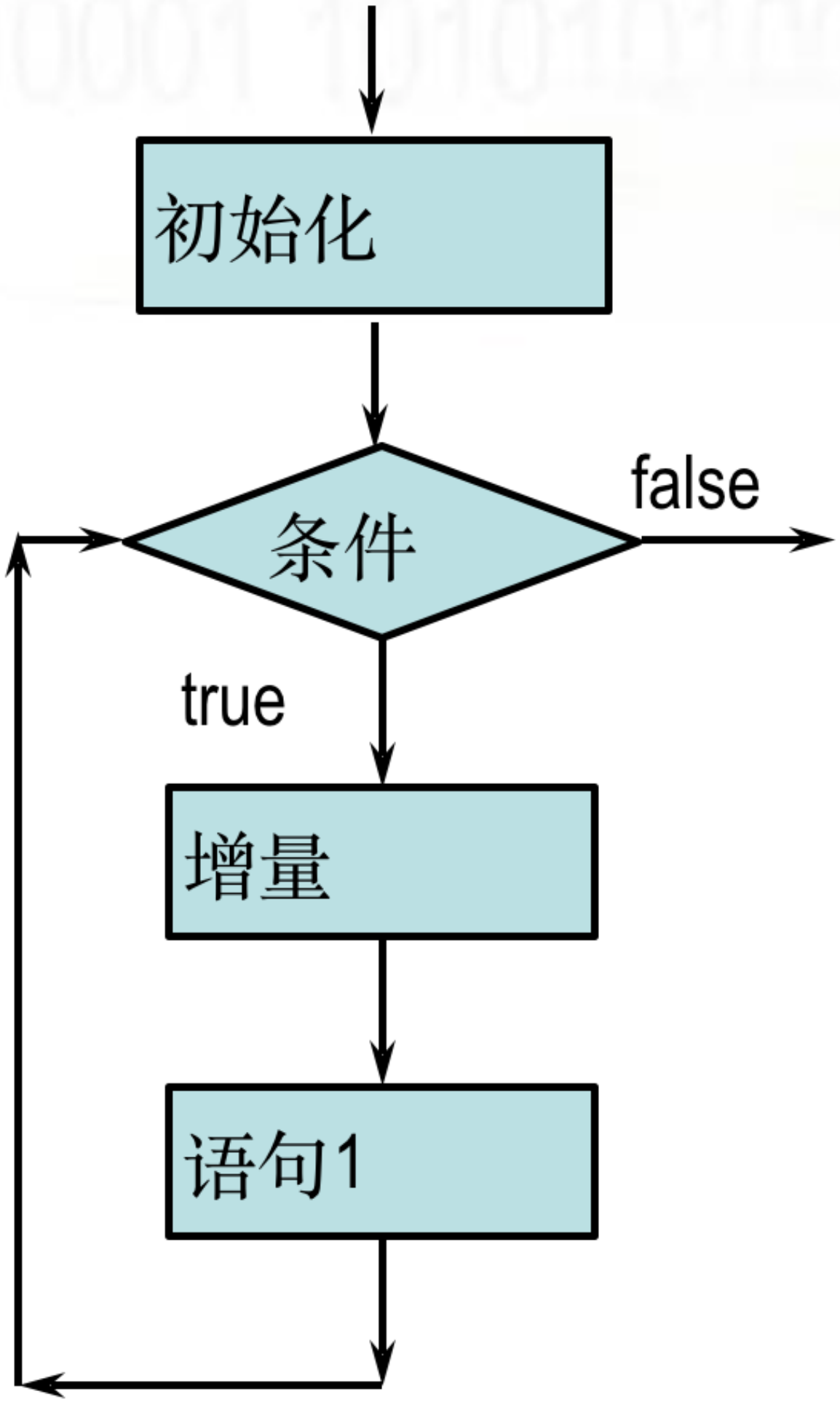



```
switch(x){case 1:y="星期一";    break;case 2:y="星期二";    break;case 3:y="星期三";    break;case 4:y="星期四";    break;case 5:y="星期五";    break;case 6:y="星期六";    break;case 7:y="星期日";    break;default: y="未定义";
}
```

switch比else if结构更加简洁清晰，使程序可读性更强,效率更高。

switch为什么效率高?

for循环基本格式for (初始化;条件;增量){   语句1;   ...}功能说明实现条件循环，当条件成立时，执行语句1，否则跳出循环体


初始化
条件
false
true
增量
语句1

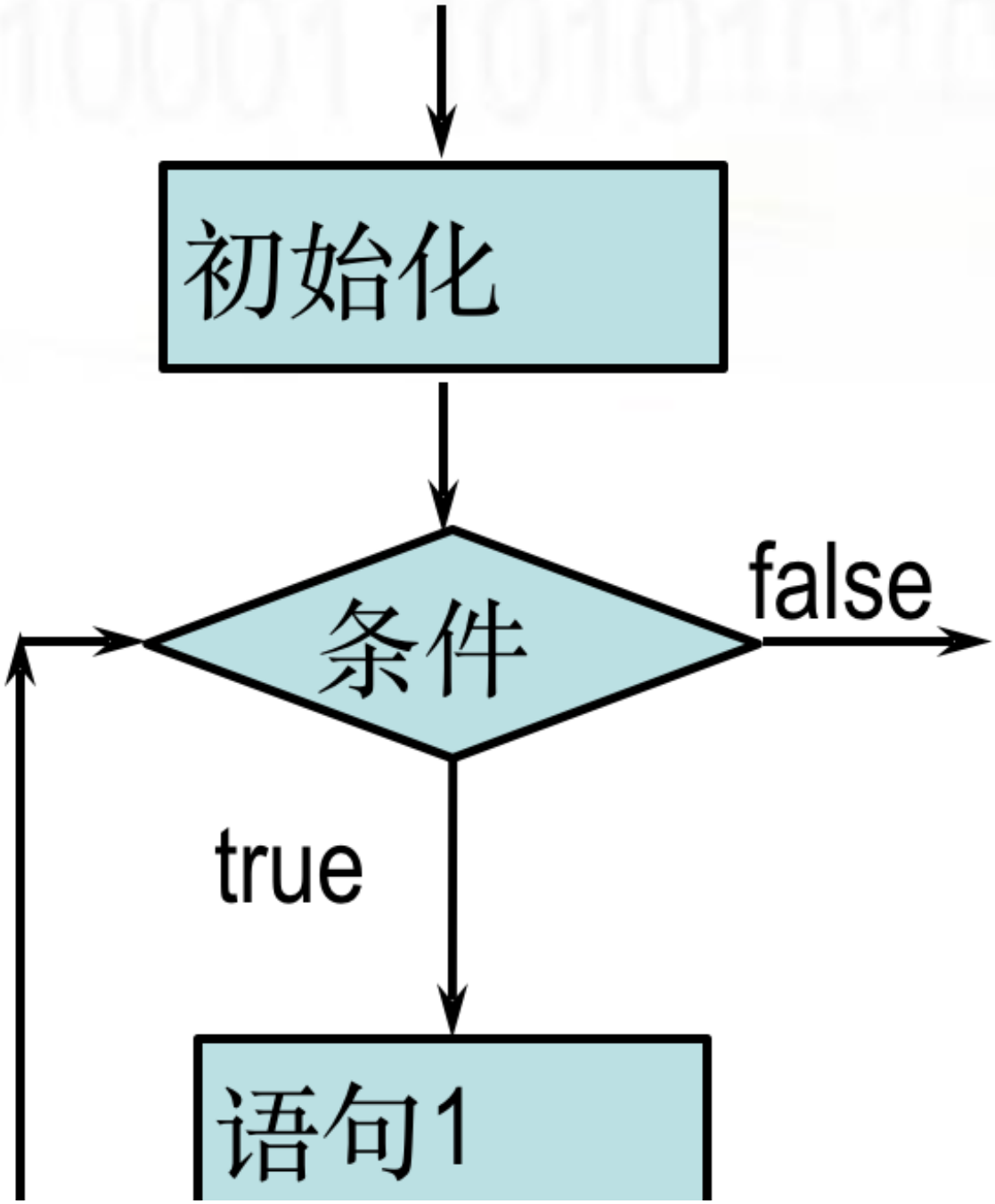
```
for (var i=1;i<=7;i++){    document.write("hello ");    document.write("
");}----------------------------------------------    var arr=
[1,"hello",true]//var dic={"1":"111"}    for (var i in arr){
console.log(i)        console.log(arr[i])    }
```

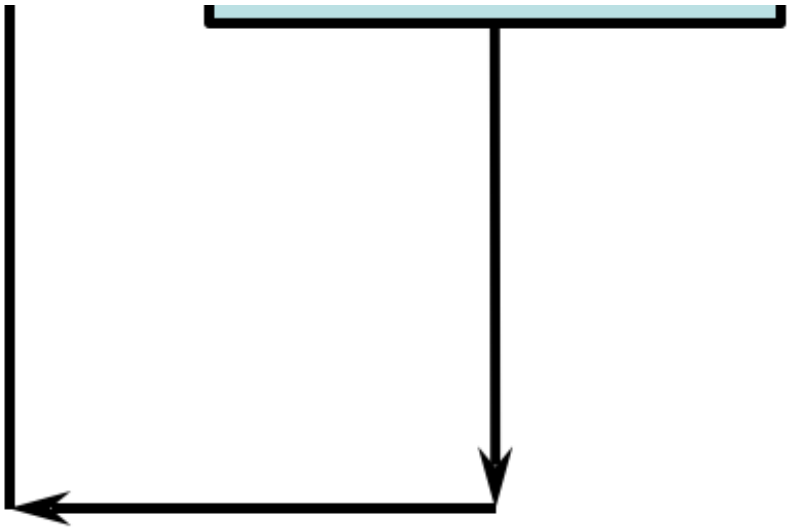
注意：

View Code

结论：for i in 不推荐使用.

while循环基本格式while （条件）{语句1；...}功能说明运行功能和for类似，当条件成立循环执行语句花括号{}内的语句，否则跳出循环





```
var i=1;while (i<=7) {    document.write("hello ");    document.write("
");    i++;}//循环输出H1到H7的字体大小
```

View Code

练习：分别用for循环和while循环计算出1-100的和？

```
try{
//这段代码从上往下运行，其中任何一个语句抛出异常该代码块就结束运行
}
catch(e)
{
// 如果try代码块中抛出了异常，catch代码块中的代码就会被执行。
//e是一个局部变量，用来指向Error对象或者其他抛出的对象
}
finally {
//无论try中代码是否有异常抛出（甚至是try代码块中有return
```

	语句），finally代码块中始终会被执行。}

注：主动抛出异常 throw Error('xxxx')

从传统意义上来说，ECMAScript 并不真正具有类。事实上，除了说明不存在类，在 ECMA-262 中根本没有出现“类”这个词。ECMAScript 定义了“对象定义”，逻辑上等价于其他程序设计语言中的类。

var o = new Object();

- 由ECMAScript定义的本地对象.独立于宿主环境的 ECMAScript 实现提供的对象.(native object)
- ECMAScript 实现提供的、独立于宿主环境的所有对象，在 ECMAScript 程序开始执行时出现.这意味着开发者不必明确实例化内置对象，它已被实例化了。ECMA-262 只定义了两个内置对象，即 Global 和 Math （它们也是本地对象，根据定义，每个内置对象都是本地对象）。（built-in object）
- 所有非本地对象都是宿主对象（host object），即由 ECMAScript 实现的宿主环境提供的对象。所有 BOM 和 DOM 对象都是宿主对象。

object对象：ECMAScript 中的所有对象都由这个对象继承而来；Object 对象中的所有属性和方法都会出现在其他对象中

ToString() ： 返回对象的原始字符串表示。

ValueOf() ：返回最适合该对象的原始值。对于许多对象，该方法返回的值都与 ToString() 的返回值相同。

11种内置对象

包括：

Array ,String , Date, Math, Boolean, Number Function, Global, Error, RegExp , Object

简介：

在JavaScript中除了null和undefined以外其他的数据类型都被定义成了对象，也可以用创建对象的方法定义变量，String、Math、Array、Date、RegExp都是JavaScript中重要的内置对象，在JavaScript程序大多数功能都是通过对象实现的

```
aa=Number.MAX_VALUE; //利用数字对象获取可表示最大数var bb=new String("hello JavaScript"); //创建字符串对象var cc=new Date();//创建日期对象var dd=new Array("星期一","星期二","星期三","星期四"); //数组对象
```

内置对象的分类

类型	内置对象	介绍
数据对象	Number	数字对象
	String	字符串对象
	Boolean	布尔值对象
组合对象	Array	数组对象
	Math	数学对象
	Date	日期对象
高级对象	Object	自定义对象
	Error	错误对象
	Function	函数对象
	RegExp	正则表达式对象
	Global	全局对象

自动创建字符串对象：

```
var str1="hello world";
alert(str1.length);
alert(str1.substr(1,5));
```

调用字符串的对象属性或方法时自动创建对象，用完就丢弃

手工创建字符串对象

```
var str1=new String("hello word");
alert(str1.length);
alert(str1.substr(
```

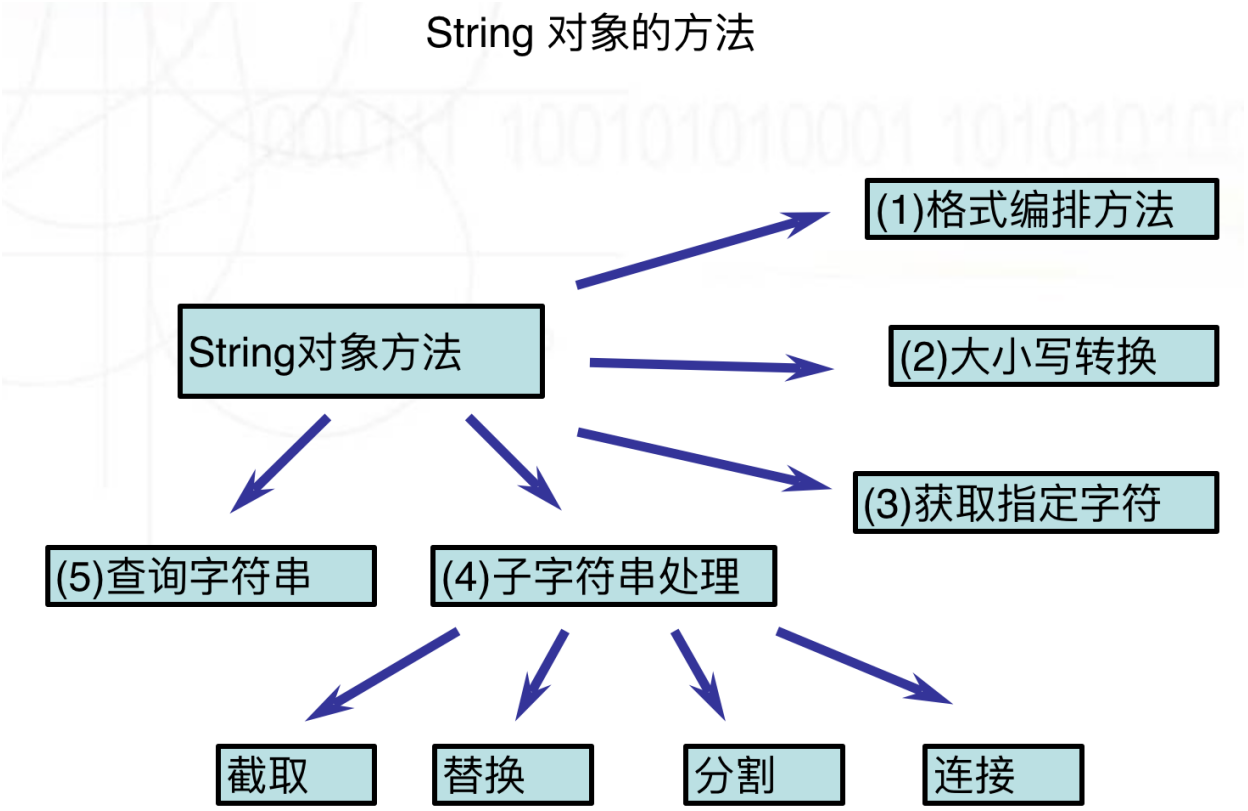
	1,3));

采用new创建字符串对象str1，全局有效

String对象的属性

1 2	获取字符串长度 length

```
var str1="String对象";
var str2="";
alert("str1长度 "+str1.length);
alert("str2长度 "+str2.length);
```



方 法	返 回 值	说 明
fontcolor(“color”)	 string	返回字体颜色定义字符串，color参数可以为red、blue、green等
fontsize(“size”)	 string	返回字体大小定义字符串，size从小到大可以定义为1到7
link(“url”)	 string	返回超链接定义字符串，url为网络超链接

类 别	方 法	说 明
大小写转换	toLowerCase()	返回小写字符串
	toUpperCase()	返回大写字符串
获取指定字符	charAt(index)	返回指定位置字符
	charCodeAt(index)	返回指定位置字符Unicode编码
查询字符串	indexOf(findstr,index)	返回正向的索引位置
	lastIndexOf(findstr)	返回反向的索引位置
	match(regexp)	返回匹配的字符串
	search(regexp)	返回找到字符串的首字符索引
类 别	方 法	说 明
截取子字符串	substr(start,length)	返回从索引位置start开始长为length的子字符串
	substring(start,end)	返回start开始end结束的子字符串
	slice(start,end)	同substring，但允许使用负数表示从后计算位置
替换子字符串	replace(findstr,tostr)	返回替换finstr为tostr之后的字符串
分割字符串	split(bystr)	返回由bystr分割成的字符串数组
连接字符串	concat(string)	返回与string连接后的字符串

String对象的方法(1) —— 格式编排方法

格式编排方法返回值列表

View Code

String对象的方法(2)——　大小写转换

```
var str1="AbcdEfgh"; var str2=str1.toLowerCase();var
str3=str1.toUpperCase();alert(str2);//结果为"abcdefgh"alert(str3);//结果
为"ABCDEFGH"
```

String对象的方法(3) —— 获取指定字符

书写格式x.charAt(index)x.charCodeAt(index)使用注解x代表字符串对象index代表字符位置index从0开始编号charAt返回index位置的字符charCodeAt返回index位置的Unicode编码----------var str1="welcome to the world of JS! 苑昊";var str2=str1.charAt(28);var str3=str1.charCodeAt(28);alert(str2);//结果为"苑"alert(str3);//结果为33489

String对象的方法(4)——　查询字符串

View Code

String对象的方法(5) ——子字符串处理

截取子字符串

View Code

替换子字符串

View Code

分割字符串

View Code

连接字符串

View Code

创建数组对象

View Code

创建二维数组

View Code

Array对象的属性

获取数组元素的个数：length

View Code

Array对象的方法

类 别	方 法	说 明
获取子数组	slice(start,end)	通过数组元素起始和结束索引号获取子数组
	splice(start, deleteCount, value, ...)	对数组指定位置进行删除和插入
进出栈操作	push(value, ...)	数组末端入栈操作
	pop()	数组末端出栈操作
	unshift(value,...)	数组首端入栈操作
	shift()	数组首端出栈操作

类 别	方 法	说 明
连接数组	join(bystr)	返回由bystr连接数组元素组成的字符串
	toString()	返回由逗号(,)连接数组元素组成的字符串
	concat(value,...)	返回添加参数中元素后的数组
数组排序	reverse()	返回反向的数组
	sort()	返回排序后的数组

连接数组-join方法

View Code

连接数组-concat方法

View Code

数组排序-reverse sort

View Code

数组切片-slice

View Code

删除子数组

View Code

数组的进出栈操作(1)

View Code

数组的进出栈操作(2)

View Code

总结js的数组特性：

View Code

创建Date对象

View Code

Date对象的方法—获取日期和时间

View Code

练习实例：

View Code

Date对象的方法—设置日期和时间

View Code

Date对象的方法—日期和时间的转换

View Code

View Code

View Code

函数的定义：

```
function 函数名 (参数){ 函数体;
return返回值;
}
```

功能说明：

- 可以使用变量、常量或表达式作为函数调用的参数
- 函数由关键字function定义
- 函数名的定义规则与标识符一致，大小写是敏感的
- 返回值必须使用return

Function 类可以表示开发者定义的任何函数。

用 Function 类直接创建函数的语法如下：

```
function 函数名 (参数){
    函数体;
   return 返回值;
}
//another way:
var 函数名 = new Function("参数1","参数n","function_body");
```

虽然由于字符串的关系，第二种形式写起来有些困难，但有助于理解函数只不过是一种引用类型，它们的行为与用 Function 类明确创建的函数行为是相同的。

实例：

```
alert(1);
functionfunc1(){
alert('hello
yuan!');
return8
}
ret=func1();
alert(ret)


- - - - - - - - - - - - - - -
varfunc1=newFunction("name","alert(\"hello\"+name);")
func1("yuan")
```

注意：js的函数加载执行与python不同，它是整体加载完才会执行，所以执行函数放在函数声明上面或下面都可以：

View Code

Function 对象的 length 属性

如前所述，函数属于引用类型，所以它们也有属性和方法。

比如，ECMAScript 定义的属性 length 声明了函数期望的参数个数。

```
alert(func1.length)
```

```
)
```

Function 对象的方法

Function 对象也有与所有对象共享的 valueOf() 方法和 toString() 方法。这两个方法返回的都是函数的源代码，在调试时尤其有用。

```
1  alert(void(fun1(1,2)))
```

运算符void()作用：拦截方法的返回值

函数的调用

View Code

函数的内置对象arguments

View Code

匿名函数

View Code

作用域

js的作用域和py相似，if while等控制语句并没有自己作用域；而函数是有自己的作用域的；

```
if(1==1){
var s=12;    }    console.log(s);//12    // ----------------------    function f(){        var temp=666;    }    f();    console.log(temp);//Uncaught ReferenceError: temp is not defined
```

嵌套函数的作用域：

例1:

```
var city = 'beijing';    function func(){        var city = 'shanghai';    function inner(){            var city = 'shenzhen';    console.log(city);        }        inner();    }    func();
```

例2:

```
var city = 'beijing';function Bar(){    console.log(city);}function func(){    var city = 'shanghai';    return Bar;}var ret = func();ret();    //beijing
```

闭包：

```
var city = 'beijing';function func(){    var city = "shanghai";    function inner(){        // var city = "langfang";        console.log(city);    }    return inner;}var ret = func();ret();
```

思考题1：

View Code

作用域链(Scope Chain)：

在JavaScript中，函数也是对象，实际上，JavaScript里一切都是对象。函数对象和其它对象一样，拥有可以通过代码访问的属性和一系列仅供JavaScript引擎访问的内

部属性。其中一个内部属性是[[Scope]]，由ECMA-262标准第三版定义，该内部属性包含了函数被创建的作用域中对象的集合，这个集合被称为函数的作用域链，它决定了哪些数据能被函数访问。

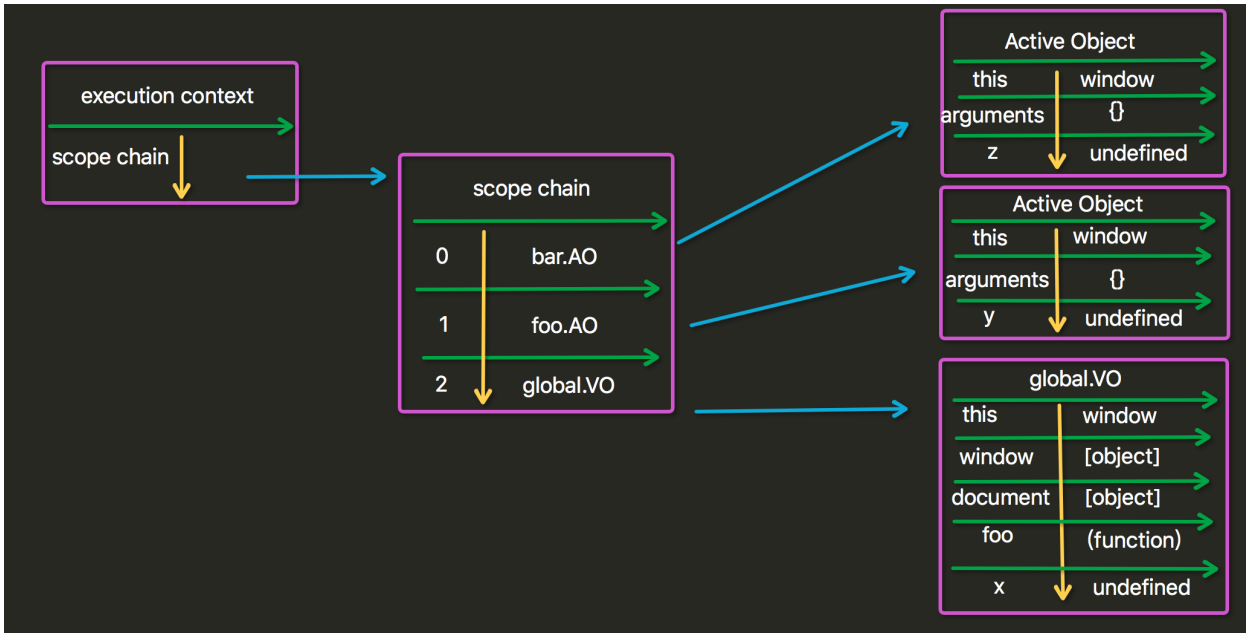
```
var x=1;function foo() {    var y = 2;        function bar() {        var z = 3;
}}#bar的作用域链:  barScopeChain=[bar.AO, foo.AO, global.VO];
#foo的作用域链:  fooScopeChain=[foo.Ao, global.VO];
View Code
```

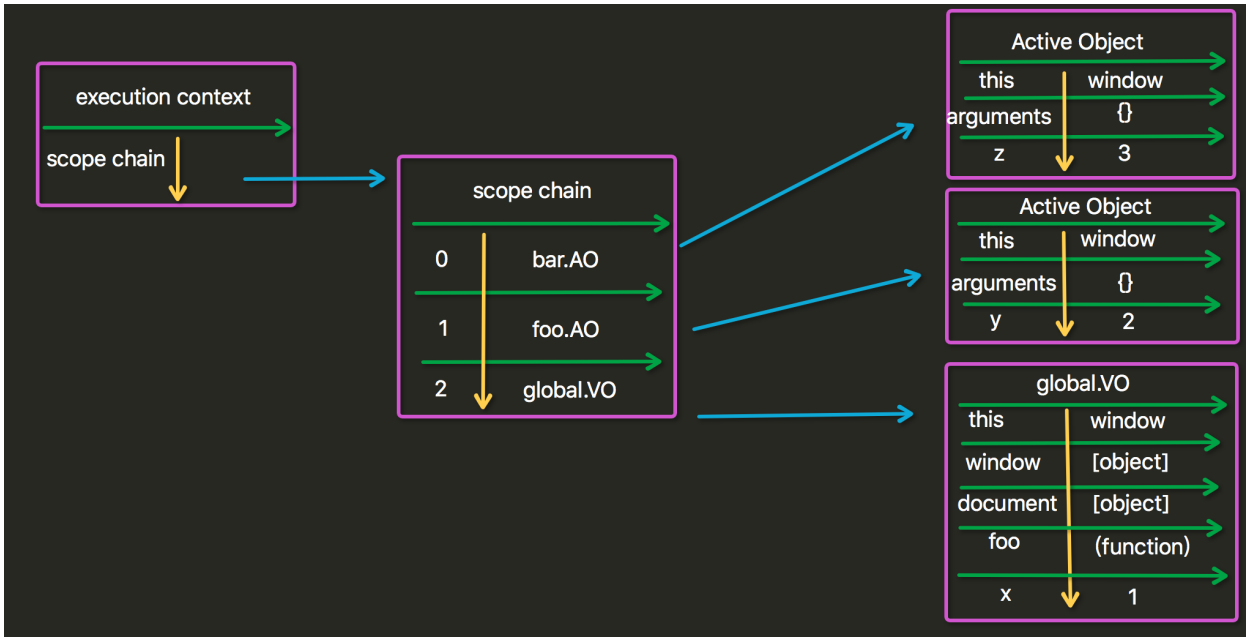
创建作用域链的过程

函数进入全局，创建VO对象，绑定x属性<入栈> global.VO={x=underfind; foo:reference of function}(这里只是预解析，为AO对象绑定声明的属性，函数执行时才会执行赋值语句，所以值是underfind) 遇到foo函数，创建foo.AO，绑定y属性<入栈> foo.AO={y=underfind, bar:reference of function} 遇到bar函数，创建bar.AO，绑定z属性<入栈> bar.AO={z:underfind} 作用域链和执行上下文都会保存在堆栈中，所以: bar函数的scope chain为: [0]bar.AO-->[1]foo.AO-->[2]global.VO foo函数的scope chain为: [0]foo.AO-->[1]global.Vo //建议：少定义全局变量//理由：因为作用域链是栈的结构，全局变量在栈底，每次访问全局变量都会遍历一次栈，//这样会影响效率

函数的scope等于自身的AO对象加上父级的scope，也可以理解为一个函数的作用域等于自身活动对象加上父级作用域.

函数执行前后的作用域链：





注意：作用域链的非自己部分在函数对象被建立（函数声明、函数表达式）的时候建立，而不需要等到执行

思考题2:

```
for (var i=1; i<=9; i++) {      setTimeout( function timer(){      console.log( i );      },1000 ); }//= = = = = = = = = = = = = = = = = = for (var i=1; i<=9; i++) {      (function(){       var j = i;       setTimeout( function timer(){      console.log( j );       }, 1000 );      })();
```

BOM（浏览器对象模型），可以对浏览器窗口进行访问和操作。使用 BOM，开发者可以移动窗口、改变状态栏中的文本以及执行其他与页面内容不直接相关的动作。

使 JavaScript 有能力与浏览器“对话”。

window对象　　所有浏览器都支持 window 对象。　　概念上讲.一个html文档对应一个window对象.

功能上讲：控制浏览器窗口的.　　使用上讲：window对象不需要创建对象,直接使用即可.

View Code

交互方法：

View Code

练习：

View Code

setInterval clearInterval

View Code

setTimeout clearTimeout

```
var ID = setTimeout(abc,2000); // 只调用一次对应函数.          clearTimeout(ID);
function abc(){        alert('aaa');     }
```

History 对象属性

History 对象包含用户（在浏览器窗口中）访问过的 URL。

History 对象是 window 对象的一部分，可通过 window.history 属性对其进行访问。

1	length 返回浏览器历史列表中的 URL 数量。

History 对象方法

back() 加载 history 列表中的前一个 URL。forward() 加载 history 列表中的下一个 URL。go() 加载 history 列表中的某个具体页面。

View Code

Location 对象包含有关当前 URL 的信息。

Location 对象是 Window 对象的一个部分，可通过 window.location 属性来访问。

Location 对象方法

location.assign(URL)location.reload()location.replace(newURL)//注意与assign的区别

回到顶部

DOM 是 W3C（万维网联盟）的标准。DOM 定义了访问 HTML 和 XML 文档的标准：

"W3C 文档对象模型（DOM）是中立于平台和语言的接口，它允许程序和脚本动态地访问和更新文档的内容、结构和样式。"

W3C DOM 标准被分为 3 个不同的部分：

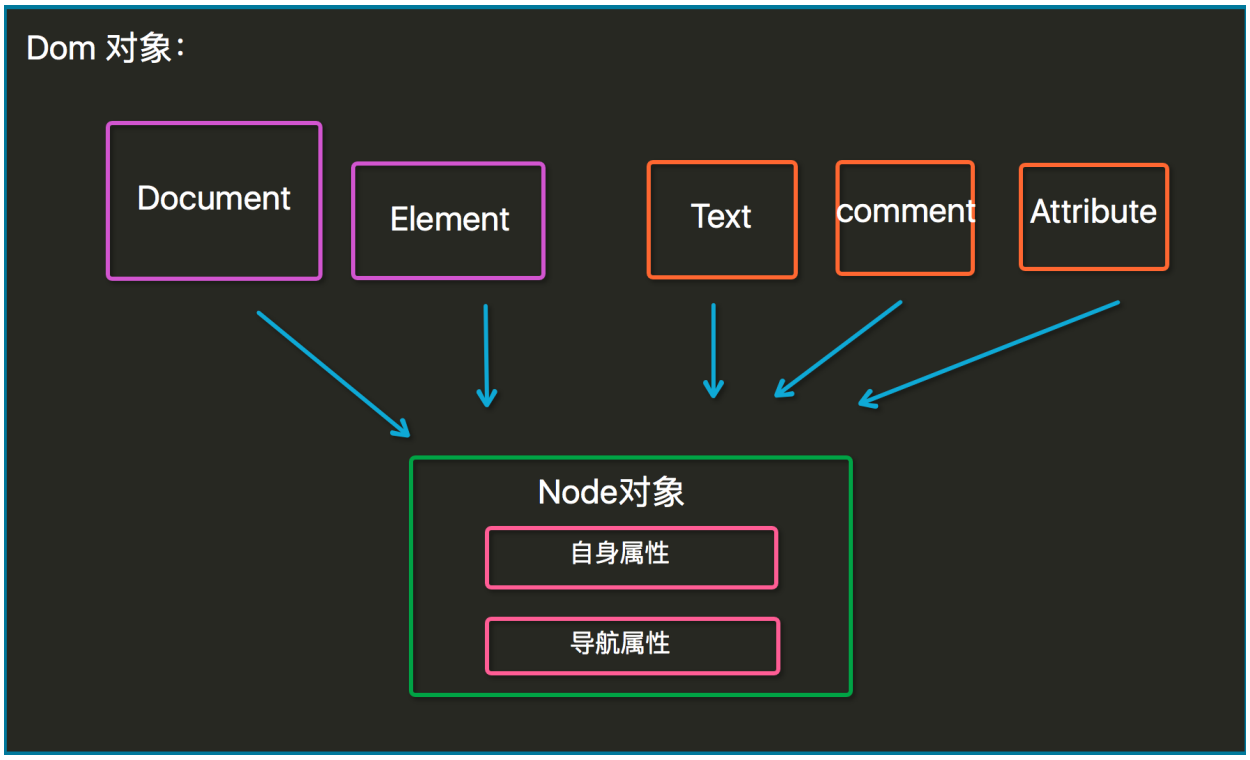
- 核心 DOM - 针对任何结构化文档的标准模型
- XML DOM - 针对 XML 文档的标准模型
- HTML DOM - 针对 HTML 文档的标准模型
- 什么是 XML DOM? - - - - >XML DOM 定义了所有 XML 元素的对象和属性，以及访问它们的方法。
- 什么是 HTML DOM? - - - - >HTML DOM 定义了所有 HTML 元素的对象和属性，以及访问它们的方法。

根据 W3C 的 HTML DOM 标准，HTML 文档中的所有内容都是节点(NODE)：

- 整个文档是一个文档节点(document对象)

- 每个 HTML 元素是元素节点(element 对象)
- HTML 元素内的文本是文本节点(text对象)
- 每个 HTML 属性是属性节点(attribute对象)
- 注释是注释节点(comment对象)

画dom树是为了展示文档中各个对象之间的关系，用于对象的导航。



节点(自身)属性:

- attributes - 节点（元素）的属性节点
- nodeType – 节点类型
- nodeValue – 节点值
- nodeName – 节点名称
- innerHTML - 节点（元素）的文本值

导航属性:

- parentNode - 节点（元素）的父节点 (推荐)
- firstChild – 节点下第一个子元素
- lastChild – 节点下最后一个子元素
- childNodes - 节点（元素）的子节点

注意：

View Code

推荐导航属性：

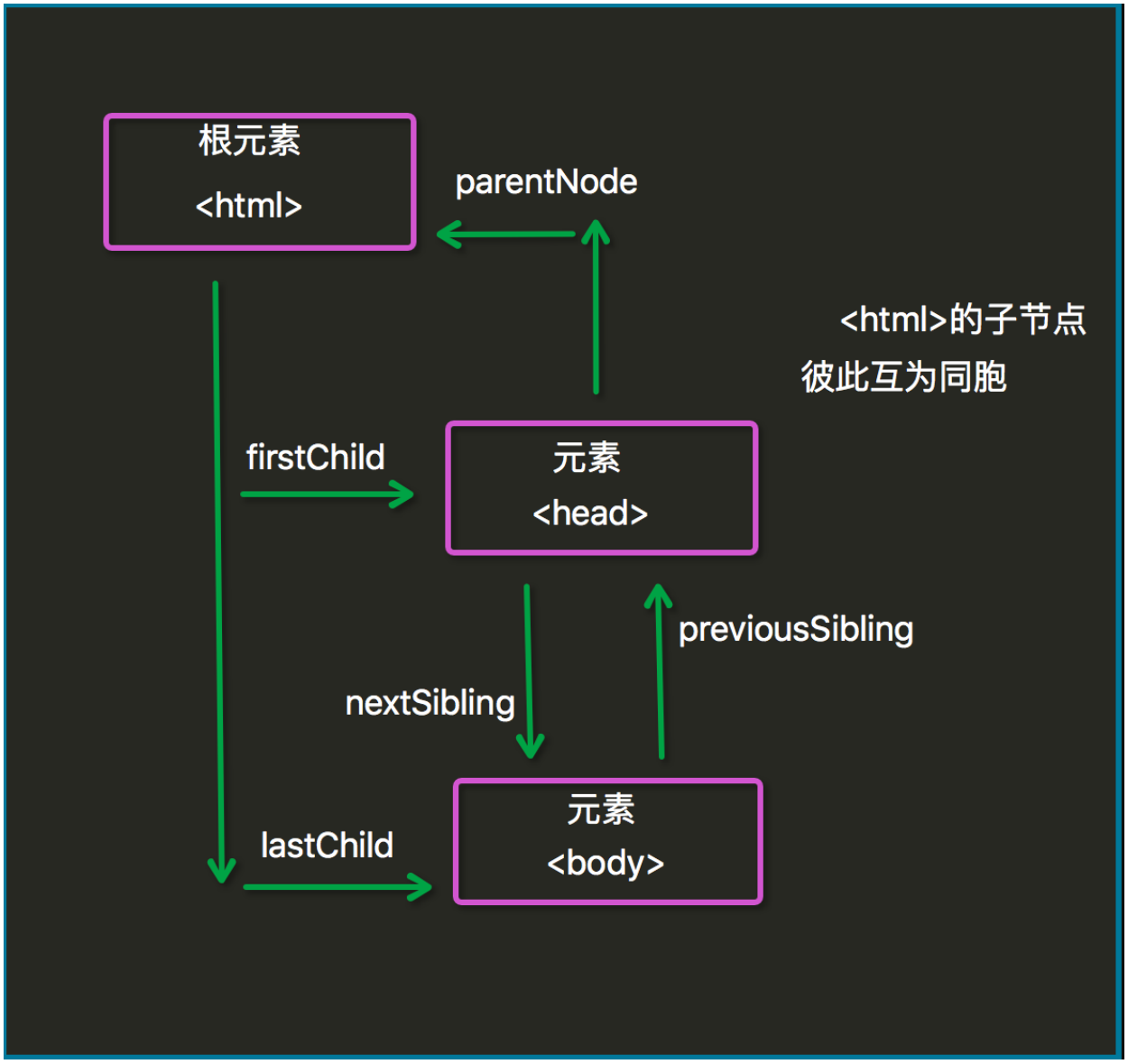
View Code

节点树中的节点彼此拥有层级关系。

父(parent),子(child)和同胞(sibling)等术语用于描述这些关系。父节点拥有子节点。同级的子节点被称为同胞（兄弟或姐妹）。

- 在节点树中，顶端节点被称为根（root）
- 每个节点都有父节点、除了根（它没有父节点）
- 一个节点可拥有任意数量的子
- 同胞是拥有相同父节点的节点

下面的图片展示了节点树的一部分，以及节点之间的关系：



访问 HTML 元素（节点）,访问 HTML 元素等同于访问节点,我们能够以不同的方式来访问 HTML 元素：

页面查找：

- 通过使用 getElementById() 方法
- 通过使用 getElementsByTagName() 方法
- 通过使用 getElementsByClassName() 方法
- 通过使用 getElementsByName() 方法

局部查找：

View Code

HTML 4.0 的新特性之一是有能力使 HTML 事件触发浏览器中的动作（action），比如当用户点击某个 HTML 元素时启动一段 JavaScript。下面是一个属性列表，这些属性可插入 HTML 标签来定义事件动作。

onclick 当用户点击某个对象时调用的事件句柄。ondblclick 当用户双击某个对象时调用的事件句柄。onfocus 元素获得焦点。 //练习：输入框onblur 元素失去焦点。 应用场景：用于表单验证,用户离开某个输入框时,代表已经输入完了,我们可以对它进行验证.onchange 域的内容被改变。 应用场景：通常用于表单元素,当元素内容被改变时触发.（三级联动）onkeydown 某个键盘按键被按下。 应用场景：当用户在最后一个输入框按下回车按键时,表单提交.onkeypress 某个键盘按键被按下并松开。onkeyup 某个键盘按键被松开。onload 一张页面或一幅图像完成加载。 onmousedown 鼠标按钮被按下。onmousemove 鼠标被移动。onmouseout 鼠标从某元素移开。onmouseover 鼠标移到某元素之上。

onmouseleave 鼠标从元素离开onselect 文本被选中。onsubmit 确认按钮被点击。

两种为元素附加事件属性的方式

View Code

注意：

View Code

onload：

onload 属性开发中 只给 body元素加.

这个属性的触发 标志着 页面内容被加载完成.

应用场景：当有些事情我们希望页面加载完立刻执行,那么可以使用该事件属性.

View Code

onsubmit:

是当表单在提交时触发. 该属性也只能给form元素使用.应用场景：在表单提交前验证用户输入是否正确.如果验证失败.在该方法中我们应该阻止表单的提交.

View Code

Event 对象

Event 对象代表事件的状态，比如事件在其中发生的元素、键盘按键的状态、鼠标的位置、鼠标按钮的状态。

事件通常与函数结合使用，函数不会在事件发生前被执行！event对象在事件发生时系统已经创建好了,并且会在事件函数被调用时传给事件函数.我们获得仅仅需要接收一下即可.

比如onkeydown,我们想知道哪个键被按下了，需要问下event对象的属性，这里就时KeyCode;

思考：onclick＝function(event){};这个方法是谁调用的?

事件传播：

View Code

增:

```
createElement(name)创建元素
appendChild();将元素添加
```

删:

```
获得要删除的元素
获得它的父元素
使用removeChild()方法删除
```

改:

第一种方式:

使用上面增和删结合完成修改

第二中方式:

使用setAttribute();方法修改属性

使用innerHTML属性修改元素的内容

查: 使用之前介绍的方法.

View Code

- 改变 HTML 内容

  改变元素内容的最简答的方法是使用 innerHTML ， innerText。

- 改变 CSS 样式

```
"p2">Hello world!
document.getElementById("p2").style.color="blue";
 .style.fontSize=48px
```

- 改变 HTML 属性

  elementNode.setAttribute(name,value)

elementNode.getAttribute(name)<--------------
>elementNode.value(DHTML)

- 创建新的 HTML 元素

  createElement(name)

- 删除已有的 HTML 元素

  elementNode.removeChild(node)

- 关于class的操作

  elementNode.className

  elementNode.classList.add

  elementNode.classList.remove

回到顶部

View Code

View Code

View Code

View Code

View Code